VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Microsoft Robotics Developer Studio - modelování úloh

Microsoft Robotics Studio - Modeling of Chosen Problems

2010

Miroslav Osmančík

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. května 2010 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli. Zvláštní poděkování patří vedoucí mé práce RNDr. Elišce Ochodkové, Ph.D., která věnovala spoustu času a energie, aby mi pomohla s řešením různých úskalí při vypracování praktické části a psání práce.

Abstrakt

Tato práce se zabývá popisem nových technologií, které se využívají k programování robota LEGO NXT. Zabýváme se multiplatformním Microsoft Robotics Developer Studio 2008 a s ním spojeným Visual Programming Language, ve kterém vznikaly úlohy popsané v této práci. Některé z úloh byly také modelovány v simulátoru robotů Visual Simulation Environment. V teoretické části je popsána základní struktura platformy, princip služeb, tvorba a práce se službami ve Visual Programming Language.

Praktická část se zaměřuje na implementaci několika modelových úloh, které byly určeny k demonstraci programování robotů. Úlohy jsou realizovány na robotovi v reálném prostředí, ale i v simulátoru. Jsou zde popsány postupy použité k implementaci a popis a řešení problémů, které se vyskytly v průběhu programování.

Klíčová slova: LEGO NXT, Microsoft Robotics Developer Studio, Visual Programming Language, robot, Visual Simulation Environment, služba, Decentralized Software Service, Concurrency and Coordination Runtime

Abstract

This thesis deals with a description of new technologies which are used to program a LEGO NXT robot. It concentrates on the multiplatform Microsoft Robotics Developer Studio 2008 and its Visual Programming Language which was used to implement the tasks described in this thesis. Some of the tasks were simulated in the product part called the Visual Simulation Environment. The basic structure of a platform, the system of services, creation and working with services in the Visual Programming Language are described in the theoretical part of the thesis.

The practical part is focused on the implementation of a few tasks which are intended to show programming robots. Our tasks are implemented not only on a real robot but even in the simulation environment. There are described procedures used for the implementation, and a description and a solution of the problems which emerged during programming.

Keywords: LEGO NXT, Microsoft Robotics Developer Studio, Visual Programming Language, robot, Visual Simulation Environment, service, Decentralized Software Service, Concurrency and Coordination Runtime

Seznam použitých zkratek a symbolů

CCR	–	Concurrency and Coordination Runtime
DSS	–	Decentralized Software Services
DSSP	–	Decentralized Software Services Protocol
HTTP	–	Hypertext Transfer Protocol
LED	–	Light-Emitting Diode
LCD	–	Liquid Crystal Display
MRDS	–	Microsoft Robotics Developer Studio
PC	–	Personal Computer - počítač
PDA	–	Personal Digital Assistant - digitální asistent
RAM	–	Random Access Memory
SOAP	–	Simple Object Access Protocol
URI	–	Unified Resource Identifier
VPL	–	Visual Programming Language
VSE	–	Visual Simulation Environment
XML	–	Extended Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Práce se zaměřuje na oblast robotiky. V moderní době je to dynamicky rozvíjející se obor, který se čím dál více dostává do povědomí veřejnosti. Automatická sekačka na trávu, automatický vysavač nebo kuchyňský robot jsou jasnými příklady toho, že i roboti mají místo v běžném životě. Nicméně roboty můžeme potkat i v oblastech zdravotnictví při důležitých chirurgických operacích, transplantacích apod., kde je potřeba přesnost zákroku taková, které není člověk sám o sobě schopen dosáhnout. Robotika hraje také významnou roli ve vesmírných nebo armádních misích, kde jsou roboti vysíláni do neprobádaných nebo nebezpečných oblastí, aby nebyl zbytečně ohrožen lidský život.

V práci se zaměřujeme na jeden z nástrojů pro programování robotů - Microsoft Robotics Developer Studia a jeho grafický nástroj Visual Programming Language. K praktickým úkolům a implementaci byl použit robot LEGO Mindstorms NXT.

Hlavním cílem práce je vytvoření několika úloh, na kterých je domonstrována práce s robotem a jeho senzory. Většina z nich je řešena na robotovi v reálném prostředí, nicméně jsme využili i možnost simulátoru, který MRDS nabízí.

Úlohy jsou v základě jednoduché. Měly by jednoduše prezentovat práci na moderním a aktraktivním robotovi. Mezi navržené úlohy patří vyjetí robota na nakloněnou rovinu, nalezení překážky a její objetí, jízda po čáře a nalezení nejkratší cestu průchodu labyrintem.

Druhá kapitola se zabývá obecným popisem platformy MRDS, principy a službami. Jsou zde vysvětleny základní technologie DSS a CCR, funkce služeb a manifestů. Popisuje také simulační prostředí a práci v něm.

V třetí kapitole se práce zaměřuje konkrétně na popis stavebnice LEGO NXT robota. Jsou zde předvedeny všechny základní senzory, jejich vlastnosti a způsob použití. Kapitola vysvětluje možnosti komunikace robota s jiným robotem nebo počítačem a její uskalí. Jak název LEGO napovídá, modul robota je možno různě modifikovat. V závěru je prezentována závěrečná konstrukce robota, se kterou se modelové úlohy realizovaly.

V další kapitole se seznámíme s grafickým prostředím pro programování - Visual Programming Language. Jsou zde ukázány základní úkony, jak toto prostředí používat, popis nástroje a způsob jak program zkopilovat do kódu C#.

Předposlední část popisuje konkrétní modelované úlohy. Ke každé úloze je uveden její návrh, postup, jakým byla řešena a následně také problémy, se kterými jsme se setkali. U některých úloh uvádíme konkrétní postupy při konfiguraci MRDS a tipy pro práci v něm.

Poslední kapitola je věnována poslední modelové úloze - průchod robota labyrintem. Práce popisuje návrh a postup řešení. Opět se seznámíme s různými obtížemi, které nastaly při implementaci. Této úloze byla věnována samostatná kapitola, neboť je postavena na základě předcházejících úkolů. Robot musí projet bludiště sestavené z čar ve čtvercové síti, prozkoumat ho a zapsat si všechny možné cesty. Následně by měl najít nejkratší cestu průchodem bludiště.

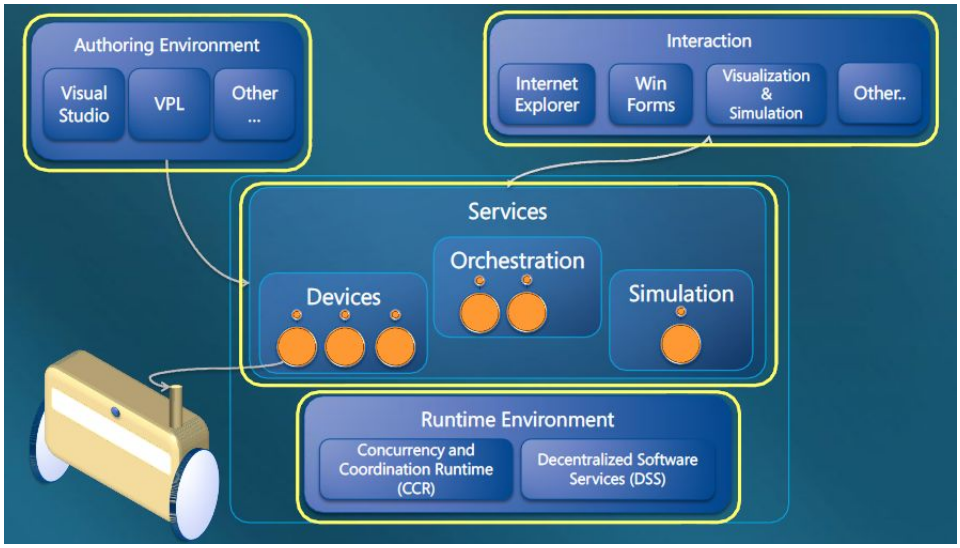
2 Microsoft Robotics Developer Studio

2.1 Architektura

MRDS je založeno na platformě .NET čili většina programů je psána v jazyce C# nebo Visual Basic .NET. S MRDS jste schopni programovat různé typy robotů pomocí obdobných kódů. MRDS obsahuje run-time prostředí, ve kterém je možné psát asynchronní a distribuované aplikace. Komponenty run-time prostředí jsou Concurrency and Coordination Runtime a Decentralized Software Services. Obecná architektura je uvedena na obrázku 1.

MRDS nabízí také Visual Simulation Environment, ve kterém můžeme modelovat ve 3D různé situace s robotem, ikdyž ho reálně ještě nevlastníme. VSE plně podporuje fyziku prostředí, proto je možné simulovat sílu, hmotnost, barvu, velikost, světlo a další prvky reálné situace.

VPL je vizuální nástroj pomocí kterého lze programovat roboty a vytvářet služby pomocí grafických prvků na principu drag and drop. Programování je názornější a lépe se v něm lze zorientovat. Vizuální program lze zkompilovat jako službu do kódu MRDS.



Obrázek 1: Architektura MRDS [6]

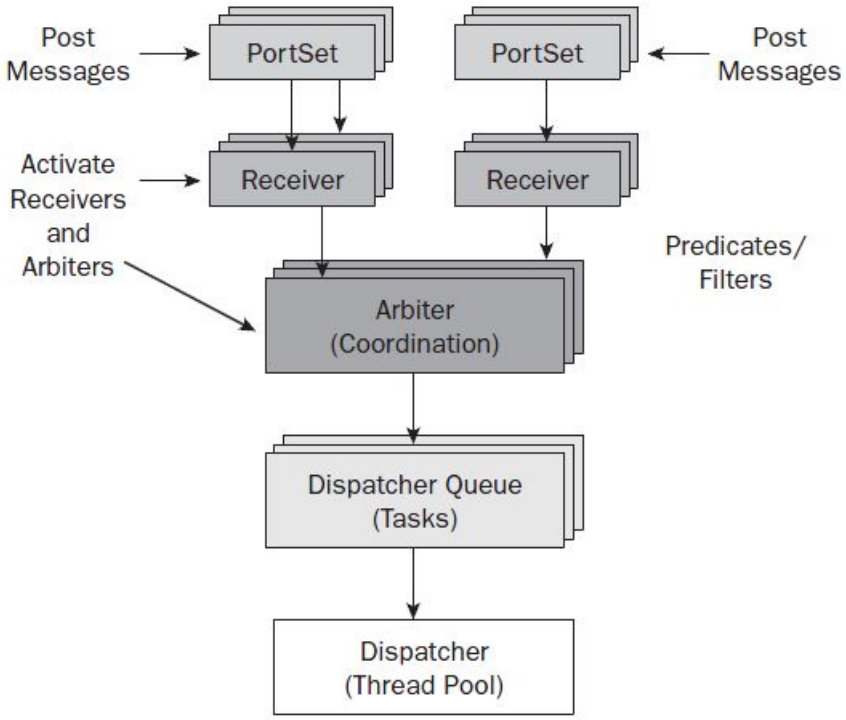
2.2 Concurrency and Coordination Runtime

CCR je knihovna na platformě .NET a její třídy a metody jsou k dispozici vývojářům, kterým zjednodušují práci při psaní vícevláknových kódů. CCR za ně ovládá a řídí souběh operací, koordinaci a chrání před asynchronní kolizí.

Když přijde zpráva, je umístěna do fronty nazvané *port*. Porty jsou fronty zpráv, které přijímají zprávy stejného datového typu. Zpráva je v portu dokud ji nezpracuje

tzv. *receiver*. Každý segment kódu může běžet asynchronně. Programování robotů stojí na asynchronních operacích. Jinak bychom nemohli ovládat například senzor a zároveň jízdu robota. Asynchronnost umožňuje spouštět a dokončovat služby v různých časech a nezávisle na sobě.

Pokud je třeba počkat, než se některá z operací dokončí, pak to řídí CCR. Používá k tomu tzv. *dispatchery*, které rozhodují o tom, který segment kódu aktuálně běží. Může nastat situace, že je třeba vytvořit podmínku mezi dvěma porty. Vytvoří se logický výraz jako je *Join* (čeká než přijdou všechny vstupní zprávy a tvoří logický AND) nebo *Choice* (čeká na alespoň jednu zprávu a tvoří logický OR). O vyhodnocení těchto podmínek se stará tzv. *arbiter*. Obrázek číslo 2 vyobrazuje stručný přehled architektury Concurrency and Coordination Runtime.



Obrázek 2: Architektura CCR [2]

2.3 Decentralized Software Service

DSS je knihovna, která je grafickou nadstavbou pro CCR. Aplikace postavená na DSS je složena z několika nezávislých služeb, které mohou běžet paralelně. DSS je zodpovědné za řízení toku zpráv mezi službami. Načítá konfiguraci služeb, kontroluje bezpečnost, řídí přístup k lokálním souborům a poskytuje uživatelské webové rozhraní pro vývojáře přes DSS uzel.

Používá svůj vlastní protokol DSS Protocol. DSSP rozlišuje stav a chování služby. Reprezentuje veškerý přístup ke službě jako operaci se stavem služby.

Abychom znali konkrétní stav sensorů a motorů robota je pro to třeba použít řídící aktivity pro chování robota. V dynamickém prostředí hrají důležitou roli asynchronní hlášení ze sensorů nazvané jako *notifikace*.

DSSP rozšiřuje protokol HTTP a nabízí modifikované metody pro operace se zprávami. Například metoda HTTP GET je přístupná v celém MRDS. Nicméně abychom mohli posílat zprávy mezi různými službami, potřebujeme je posílat ve formátu SOAP.

DSSP také nabízí několik přidaných operací, které byly vyvinuty pro potřeby MRDS. Kupříkladu podporuje registraci služby, což HTTP nepodporuje. V MRDS je častým požadavkem čekání na aktualizaci informací ze sensoru. Toto je řešeno právě registrací služby k jiné službě, čímž zajistíme, že notifikace budou službě zasílany buď v pravidelných intervalech nebo když se hodnota změní.

2.4 DSS uzel

DSS uzel je prostředí, ve kterém běží všechny služby. Musí být spuštěno ještě před tím než spustíme jakoukoliv službu. K DSS uzlu lze přistupovat přes webové rozhraní běžící na adrese http://localhost:50000. Při vstupu na toto rozhraní se setkáme s požadavkem na přihlášení, které je shodné s tím, které máte nastavené pro Windows.

Můžeme zde sledovat všechny běžící služby, debugovací zprávy, umístění běžících služeb, informace o všech dispatcherech a jejich frontách běžících v uzlu a další informace o aplikaci.

2.5 Služby

Pomocí služeb v MRDS ovládáme sensory, motory a řídící jednotku robota. Jsou to klíčové komponenty MRDS. Služba je kód, který je schopen provést nějakou operaci nad jedním nebo i více objekty. Jsou použity ke sběru dat ze sensorů robota, pro jeho řízení, komunikaci s externími aplikacemi atp. Robotická aplikace se skládá z mnoha služeb, které pracují společně se společným cílem - ovladání robota.

Každá vytvořená služba má jedinečný identifikátor URI, který ji reprezentuje v celém MRDS. Služba může mít svou partnerskou službu tak, že si navzájem posílají požadavky a odpovědi.

2.6 Manifest

Manifest je XML soubor. Obsahuje seznam služeb a jejich partnerských služeb, které mají být spuštěny společně. Může v něm být i konfigurace, ale to záleží na potřebách služby. Manifest soubor používá schéma, které je definováno firmou Microsoft. Základní sadu manifestů pro různé služby nabízí MRDS již po instalaci. Jsou umístěny v `C:\users\..\Microsoft Robotics Dev Studio 2008\samples\Config`. Jsou zde také umístěny konfigurační manifesty pro hardware robotů - pro senzory, řídící kostku a další.

Vzor manifestu *New.manifest.xml*, který vytvoří MRDS automaticky s novým projektem je demonstrován ve Výpisu č.1. Manifesty lez upravovat přímo v XML kódu pomocí jakéhokoliv textového editoru nebo také pomocí nástroje v MRDS nazvaného *DSS Manifest Editor*. Ten je nápadně podobný VPL a lze zde graficky upravovat partnerství a konfiguraci jednotlivých služeb aplikace.

```
<?xml version="1.0" ?>
<Manifest
    xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html"
    xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html"
    >
    <CreateServiceList>
        <ServiceRecordType>
            <dssp:Contract>http://schemas.tempuri.org/2010/03/new.html</dssp:Contract>
        </ServiceRecordType>
    </CreateServiceList>
</Manifest>
```

Výpis 1: Základní manifest v C#

2.7 Simulační prostředí

MRDS nabízí Visual Simulation Environment - nástroj pro vytváření 3D simulací s robotem. VSE dodržuje reálné vlastnosti objektů, včetně hmotnosti, síly, gravitace a většiny fyzikálních zákonů. Výhodou simulátoru je možnost pracovat s roboty, modelovat jejich chování i v případě, že žádného reálného nevlastníme. Je tak otevřena možnost vývoje a debugování algoritmů na modelu robota. Naopak nevýhodou je, že ne všechny senzory a komponenty robotů jsou v simulátoru již vytvořeny. Ve verzi MRDS 2008 nejsou graficky podpořeny všechny senzory pro robota LEGO NXT. Po základní instalaci je k dispozici jen senzor dotyku, řídící kostka a motory. Na obrázku č. 3 je ukázka VSE s LEGO robotem NXT.

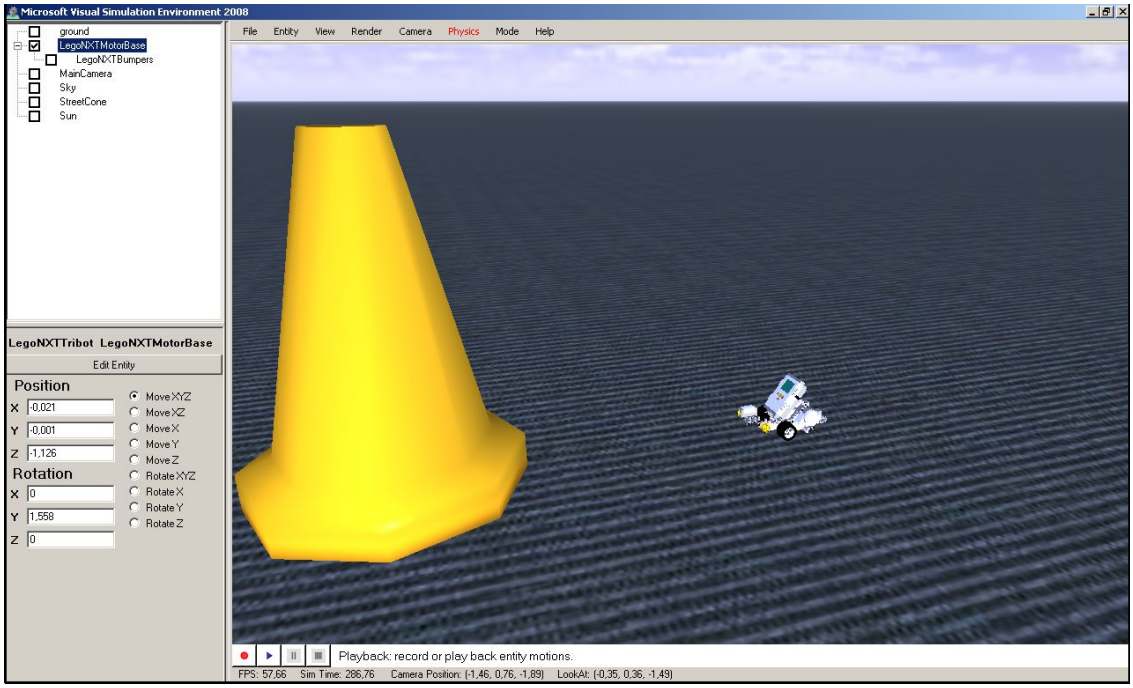
MRDS používá pro VSE AGEIA PhysX Technology engine [12], který zajišťuje celkovou fyziku simulátoru. Simuláční prostředí je založeno na platformě Microsoft XNA, která je běžně používána pro programování her pro Windows.

Ovladání simulátoru je intuitivní. Pohyb pohledu na situaci je pomocí klávesnice - písmena Q a E (nahoru a dolů), W a S (přiblížení a oddálení), A a D (vlevo a vpravo). Při označení entity v nabídce a stisknutí CTRL se entita zvyrazní. Toto pomáhá při složitějších modelech, kdy jsou objekty složeny z více entit a my chceme nastavit vlastnosti jen některé z nich.

2.7.1 Status Bar a Playback Bar

V menu View VSE si lze zaškrtnout viditelnosti těchto dvou panelů. VSE používá pravotočivý souřadný systém (x,y,z) podle obrázku č.5 k určení pozice objektu.

Status Bar ukazuje aktuální pozici kamery, což jsou prakticky naše oči. Tyto souřadnice se nastavují v manifestu před spuštěním aplikace. Dále poskytuje souřadnice *LookAt*,

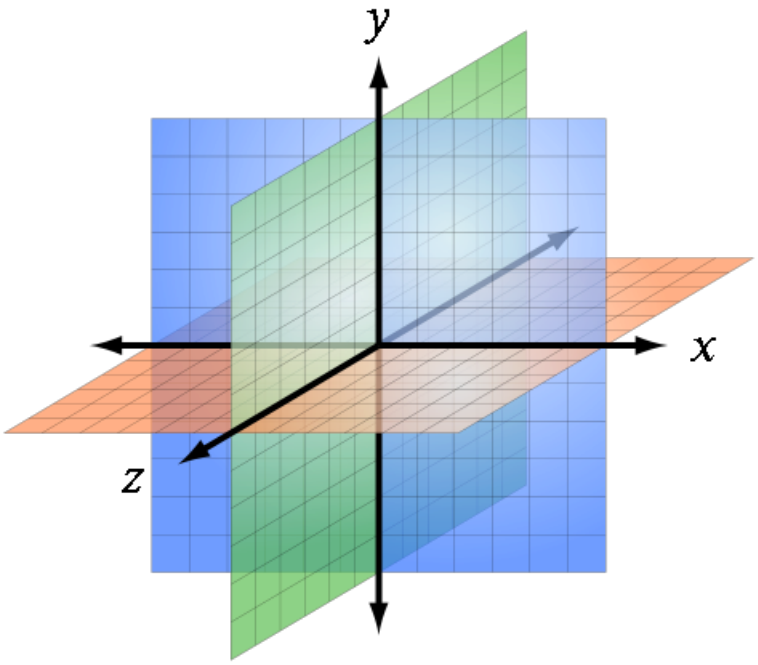


Obrázek 3: Visual Simulation Environment

které udávají místo, na které se bude kamera po spuštění centrována. Jakmile se v simulátoru pohybujeme, tak se nám souřadnice mění s konkrétním pohledem.

Playback Bar je šikovný nástroj pomocí kterého můžeme nahrávat činnost robota v modelovaných situacích a následně porovnávat jeho různé aspekty. Umožňuje i zpětné přehrání videa. Po stisku tlačítka se symbolem play na liště si v dialogové okně stačí jen vybrat soubor s příponou *.plb*. Obě lišty jsou na obrázku č. 4.

Obrázek 4: Status Bar a Playback Bar



Obrázek 5: Pravotočivý souřadnicový systém VSE [7]

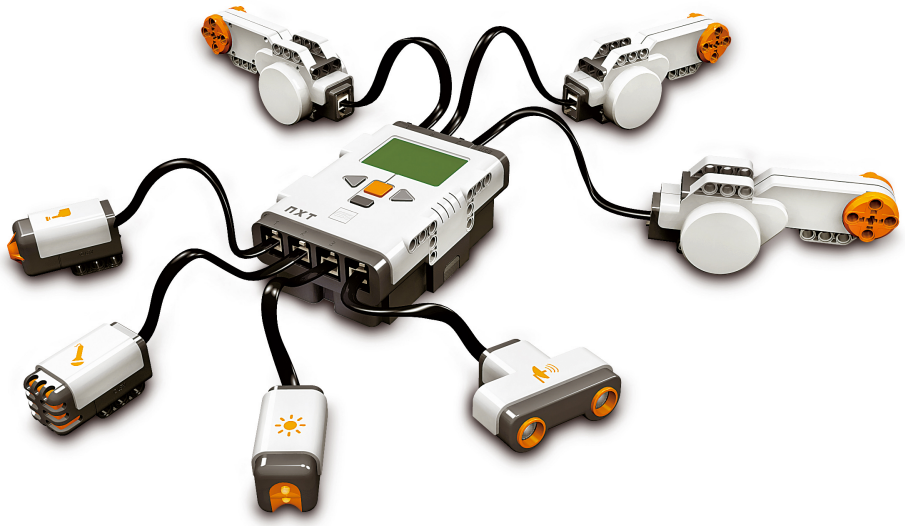
3 Robot LEGO NXT

3.1 Popis stavebnice

LEGO Mindstorm NXT je stavebnice, ze které lze sestavovat různé podoby robota. Obsahuje hlavní řídící jednotku, sensory, servo motory, kabely a další propriety k modelování úloh (např. pole s barvami a čarou, balónky ...). Komunikace robota s PC se liší od staršího modelu. Infračervené záření je nahrazeno technologií bluetooth nebo kabelem USB 2.0. Tento kabel je také součástí základní stavebnice.

V sadě s robotem je dodáván také jednoduchý software pro programování a ovládání LEGO NXT robota, tzv. *LEGO MINDSTORMS Education NXT Software v1.1*. Jedná se o grafický programovací nástroj. Programovací jazyk v tomto softwaru se jmenuje NXT-G. Programy vytvořené v NXT-G můžeme nahrát přímo do řídící kostky a spouštět je přímo na robotovi bez spojení s počítačem. Tento software je určen jen pro robota LEGO NXT, čili oproti MRDS není multiplatformní.

3.2 Hardwarové komponenty robota NXT



Obrázek 6: NXT Řídící jednotka a senzory [8]

3.2.1 Řídící jednotka

Řídící kostka je hlavní částí stavebnice. Představuje jakýsi mozek robota, díky němuž je robot schopem vykonávat operace a úkoly, které mu program nařídí. Kostka zpracovává všechny údaje z připojených senzorů a případně je posílá zpět aplikaci. Řídící jednotka je 32-bit mikroprocesor s 256 KB Flash pamětí a 64 KB RAM pamětí. Jednotka dále nabízí

adaptér bluetooth třídy II, USB port (12 MBit/s), LCD maticový displej s rozměry 100 x 64 pixelů a reproduktor frekvencí 8kHz. Na obrázku číslo 6 můžete vidět řídící jednotku se všemi zapojenými porty (senzory a motory).

3.2.2 Dotykový senzor

Dotykový senzor znázorněný na obrázku číslo 7 poskytuje robotovi jeden ze základních smyslů - hmat. Robot pomocí něj může rozeznat, zda narazí na překážku nebo pokud chce něco uchopit, zda je dané těleso dostatečně blízko. Senzor má dva stavy - stisknutý a uvolněný. Stiskem senzoru může být vyvolána i další akce jako je mluvení, otočení, úchop aj.

Obrázek 7: Dotykový senzor [9]

3.2.3 Zvukový senzor

Na obrázku číslo 8 je senzor, který můžeme použít například pokud potřebujeme vyvolání akce robota po zatleskání nebo pokud bychom chtěli, aby při dané zvukové frekvenci něco řekl.

Zvukový senzor je analogový a reaguje na zvuky tiché, ale i hlasité. Pracuje s intenzitou zvuku v decibelech (dB) a akustických decibelech (dBA)[1]. Senzor je schopen měřit až do 90dB, což je přibližně hladina zvuku sekačky na trávu. Hodnota přijímaného zvuku není na robotovi prezentována v decibelech, ale v procentech. Pro představu - přibližná hodnota 4-5% je ticho v pokoji, 10-30% je normální rozhovor nebo hudba v blízkosti senzoru.

3.2.4 Ultrazvukový senzor

Ultrazvukový senzor na obrázku č. 9 umožňuje robotovi společně se světelným senzorem vidět. Za použití tohoto senzoru můžeme měřit vzdálenost různých překážek a následně se jim vyhnout. Senzor využívá stejného principu jako je echolokace u netopýrů.

Echolokace je postup, kdy se vysílaný zvuk od předmětu odrazí zpět do místa vysílání, kde je zpětně zachycen. Z celkového času, který uplyne od okamžiku vyslání zvukové

[1] dBA - citlivost senzoru pro zvuky slyšitelné lidským uchem; dB - citlivot senzoru pro zvuky příliš nízkých nebo vysokých frekvencí, které nejsou pro člověka rozpoznatelné

Obrázek 8: Zvukový senzor [9]

vlny (obvykle vysokofrekvenčního zvuku) do okamžiku zpětného příjmu odražené vlny (ozvěny neboli echa), se dá spočítat vzdálenost alokovaného předmětu. Tento princip využívají některé specializované elektronické přístroje, například sonary. Princip echolokace je využiván pro měření hloubky moře [10].

Senzor vysílá zvuk s frekvencí 40kHz. Čidlo je schopno určit vzdálenost od 0 do 255 centrimetrů s přesností +/- 3 cm. Je třeba brát v úvahu, že ve stejném prostoru nesmí být použito více než jedno toto čidlo. A to z toho důvodu, že by se mohly vysílané zvukové vlny navzájem rušit.

Obrázek 9: Ultrazvukový senzor [9]

3.2.5 Světelný senzor

Je to druhý senzor, po ultrazvukovém, který umožňuje robotovi realizovat vidění. Analogový světelný senzor (na obr. č.10) je schopen rozlišit obklopující a odrážené světlo. V pasivním módu na senzoru nesvítí LED a je schopen určit intenzitu světla v okolí(rozlišuje světlo a tmu). V aktivním režimu se rozvítí LED, která emituje světlo k povrchu a podle odraženého množství světla se určuje barevnost povrchu. Tato fuknce je využita při modelované úloze - Jízda po čáře, kde na základě odstínu, který robot vidí, se rozhoduje o pohybu.

Měření intenzity barvy je značně závislé na podmínkách reálného porstředí. Hodnoty se mění při změně okolního osvětlení místnosti, vzdálenosti senzoru od povrchu a struktury povrchu (hladký, drsný). Standardním měřítkem pro intenzitu je rozmezí procent od 0 do 100.

Obrázek 10: Světelný senzor [9]

3.2.6 Servo motory

Sada obsahuje tři servo motory (obr. číslo 11), které umožňují robotovi se pohybovat. Každý motor má zabudované rotační čidlo, které s přesností na jeden stupeň měří otočení robota. Motory mohou vyvinout relativně dobrou kombinaci rychlosti a otáčení robota.

Obrázek 11: Servo motor [9]

3.3 Komunikace s PC

Robot LEGO NXT může komunikovat s PC pomocí technologie bluetooth[2] nebo USB kabelem. Robota lze ovládat pomocí bluetooth i mobilním telefonem nebo lze spárovat i více LEGO NXT robotů.

Komunikace pomocí bluetooth je založena na principu *master and slave*. Pokud PC inicijuje spojení, stává se z něj master a robot NXT ho pouze akceptuje jako slave. I robot může být master. Mnohdy je toto výhodné při spojení více NXT robotů. NXT master může být připojen až ke třem dalším NXT slave. Princip vztahu master-slave spočívá v posílání zpráv. Jen master posílá zprávy se zdrojovými daty programu a může požadovat data, která mu slave má poslat. Pokud master pošle data ke slave, pak může, ale nemusí, požadovat potvrzení příjmu dat. Když master žádá data od slave, pak odpověď obsahuje daná data, pokud jsou k dispozici. Pokud nejsou k dispozici, odpověď obsahuje chybovou zprávu.

Zpráva, kterou posílá master, specifikuje, zda jí má slave potvrdit nebo ne. Podle toho se určuje, kdo bude v následující relaci vysílač a kdo přijímač. Rychlost přepínání bluetooth mezi vysílacím a přijímacím módem je okolo 50 ms a může docházet k nepřesnostem.

Každý NXT má svůj tzv. *mailbox*, ve kterém uchovává přijaté a odchozí zprávy. Mailbox má celkem deset míst, čili má frontu pro pět odchozích a pět přichozích zpráv. Master nepotřebuje fronty zpráv, jelikož si vždy od slave vyžádá odpovědi a hned je zpracovává. Všechny zprávy od master ke slave jsou uchovávány ve frontě u slave. Pokud na slave dojde zpráva do fronty, která je plná, pak je nejstarší zpráva ve frontě smazána bez toho, aby byla vykonána. Tato komunikace přes bluetooth není spolehlivá. Spojení funguje bez problému při krátkých zprávách nebo při malém (přiměřeném) počtu zpráv.

Tento problém lze jednoduše vyřešit tím, že NXT robot bude master a PC bude slave. Zprávy se nebudou ztrácet, protože RAM pamět počítače je udrží. Nicméně tato možnost nám vylučuje použití MRDS. MRDS je sice multiplatformní, ale program v něm vytvořený nelze nahrát do řídící jednotky robota. NXT robot může být master při použití LEGO MINDSTORM Education Software, jelikož tyto programy lze do řídící jednotky nahrát a následně použít počítač jako slave v daném bluetooth spojení.

Pokud chceme posílat data na robota NXT, pak by měl být slave. Pokud je robot slave, pak NXT firmware posílá potvrzení o každé přijaté zprávě. Při této konfiguraci je třeba předpokládat ztrátu zpráv. Podle pokusů z [11] se ztratí okolo 18-20% zpráv, nicméně každá zprávy byla doručena za méně než 5 ms.

Komunikace MRDS s robotem probíhá pouze pomocí technologie bluetooth. MDRS nepodporuje komunikace skrz USB. Při prvním navázání spojení PC s robotem je třeba je tzv. *spárovat*. Robot musí být zapnutý a musí mít aktivovány bluetooth. Párovací heslo je defaultně 1234. Po potvrzení kódu na obou stranách je spojení navázano. Windows přidělí bluetooth zařízení svůj COM port. Odchozí port je třeba v MRDS nastavit dle aktuálního zařízení. Tuto změnu provedeme v souboru `..\Microsoft Robotics Dev Studio 2008\samples\Config\LEGO.NXT.Brick.Config.xml`. V editoru změní-

[2] Bluetooth je bezdrátová komunikační technologie sloužící k bezdrátovému propojení mezi dvěma a více elektronickými zařízeními, jakými jsou například mobilní telefon, PDA, osobní počítač.[10]

me číslo portu v elementu <*serial port*>. Dále je třeba ověřit, že manifest řídící jednotky čte nastavení z námi upraveného souboru. To si ověříme v souboru `..\samples\Config\LEGO.NXT.Brick.Manifest.xml` v elementu <*dssp:Service*> musí být název konfiguračního souboru.

Při našich modelových úlohách používáme komunikaci bluetooth mezi robotem a PC tak, že robot je slave a PC je master.

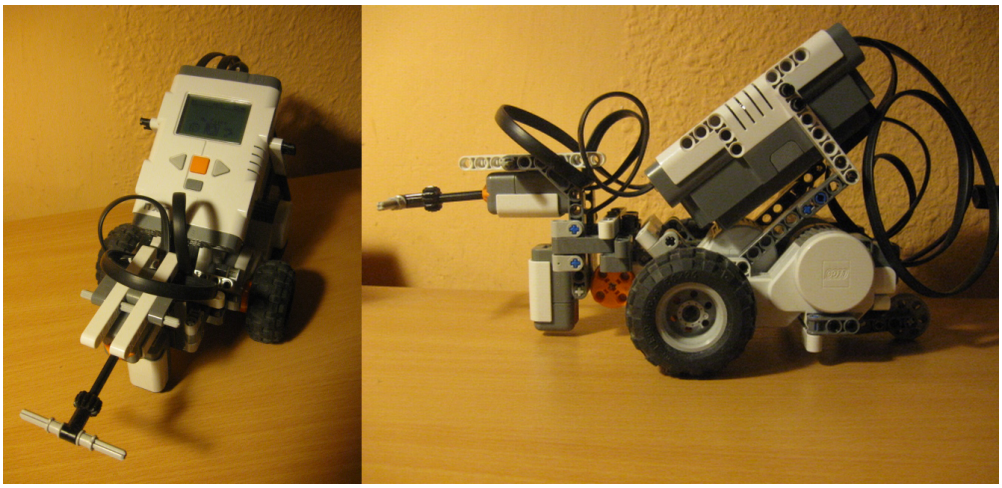
Software dodávaný v sadě s robotem - LEGO Mindstorms NXT podporuje komunikaci s robotem jak přes bluetooth, tak přes USB kabel. Přes tento software je velmi jednoduché nahrávát do robota aktuální firmware, kalibrovat senzory apod.

3.4 Konstrukce pro úlohy

LEGO NXT robot je stavebnice, ze které lze postavit spoustu různých modifikací robota dle našich představ. Návody k postavení různých typů robotů jsou v manuálu, který je součástí stavebnice.

K modelování níže popsaných úloh využijeme konstrukci, kde využíváme dotykový a světelný senzor. Základem konstrukce je tzv. NXT Tribot. Nicméně je ochuzen o ostatní čidla, která jsme u našich úloh nepotřebovali. Jak robot vypadá je zřejmé z obrázku č. 12

Parametry robota zůstávájí stejné jako u tribota. Průměr kola je 56 mm a rozteč mezi koly 11,2 cm.



Obrázek 12: Konstrukce LEGO NXT robota pro modelované úlohy

4 Visual Programming Language

4.1 Popis prostředí

VPL je součástí instalačního balíku MRDS. Je to vizuální prostředí, ve kterém lze na principu drag and drop programovat robotické aplikace. Ve VPL lze využít komponenty, které jsou ve VPL pojmenovány jako tzv. *aktivity*. Základním stavebním kamenem jsou opět služby, které mohou mít vstupní, výstupní data a notifikace. MRDS již nabízí dané spektrum služeb nebo si uživatel může vytvořit službu, kterou je možno do VPL přidat. V levém spodním sloupci je seznam služeb, které jsou k dispozici a v horním levém okně jsou aktivity - základní programovací struktury. Ve VPL je možno vytvořit vlastní aktivitu (vzdálená obdoba metody v objektově-orientovaném programovaní), u kterých lze konfigurovat vstupy, výstupy, notifikace a následně je používat v hlavním diagramu. Použité algoritmy je praktické zabalit do aktivit z důvodu přehlednosti diagramu.

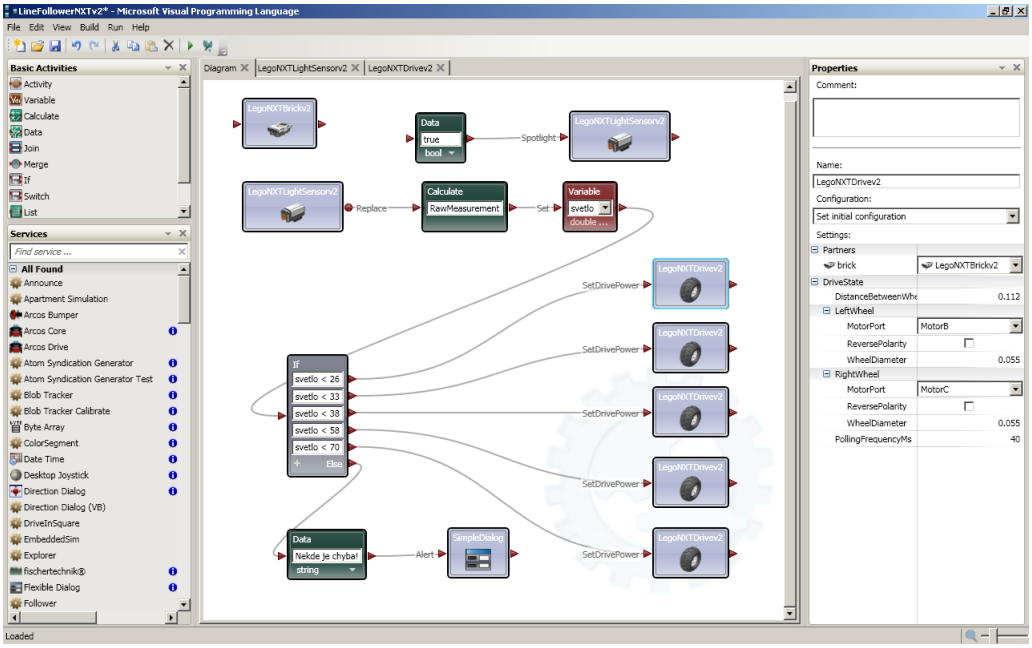
VPL poskytuje služby nejen pro NXT robota, ale i pro iRotota, FischerTechnic a další. Poskytuje také generické služby, které se používají pro programování v simulátoru MRDS nebo je možné jejich použití, pokud není daná služba pro nějaký senzor v MRDS implementována. Všechy tyto komponenty můžeme vložit do diagramu, spojovat je pomocí datových toků a tím tvořit program. Ve VPL se nepoužívají sekvence příkazů. Na základě vstupních dat se změní výstupní data. Notifikace se mění za základě změny interních dat ve službě (například změna hodnoty světelného čidla - pokud se změní hodnota, kterou čidlo registruje, pošle ji na notifikaci), takže ze služby můžeme odebírat zároveň vystupní data, ale i notifikaci.

Ve VPL je možno jednoduše programovat aplikace s paralelním a distrubuovaným chováním. Celý diagram VPL lze zkompilovat do C# kódu. Cílem VPL není naučit uživatele programovat, nesposkytuje ani přímý přístup k platformě .NET, nicméně VPL představuje rychlou a jednoduchou cestu k tvorbě robotických aplikací.

4.2 Princip implementace

Implementace programu ve VPL je na principu "chyť a táhni" (drag and drop). Do diagramu si myší přetáhneme potřebné aktivity, služby a programovací komponenty, které následně spojujeme datovými toky. Každý oddělený datový tok běží ve svém vlákně a navzájem jsou nezávislé a paralelní. Na obrázku č. 13 můžete vidět ukázku programu se třemi vlákny.

Na obrázku č. 13 je v pravém sloupci dialog Properties, který se mění podle označené komponenty nebo služby. Ve properties služeb senzorů a řídící jednotky můžeme nastavit tzv. základní konfiguraci nebo odkázat na použití manifestu jiné služby. Nastavení základní konfigurace je vidět vpravo na obrázku č. 13. Zde nastavujeme partnerskou službu (pokud nějakou chceme). U senzorů jako je světelný, ultrazvukový, motory atd. musí být partnerská služba nastavena na řídící jednotku. Dále zde nastavujeme vlastnosti konkrétního senzoru jako je například u motoru rozteč kol, průměr kola, porty a frekvence sběru dat ze senzoru.



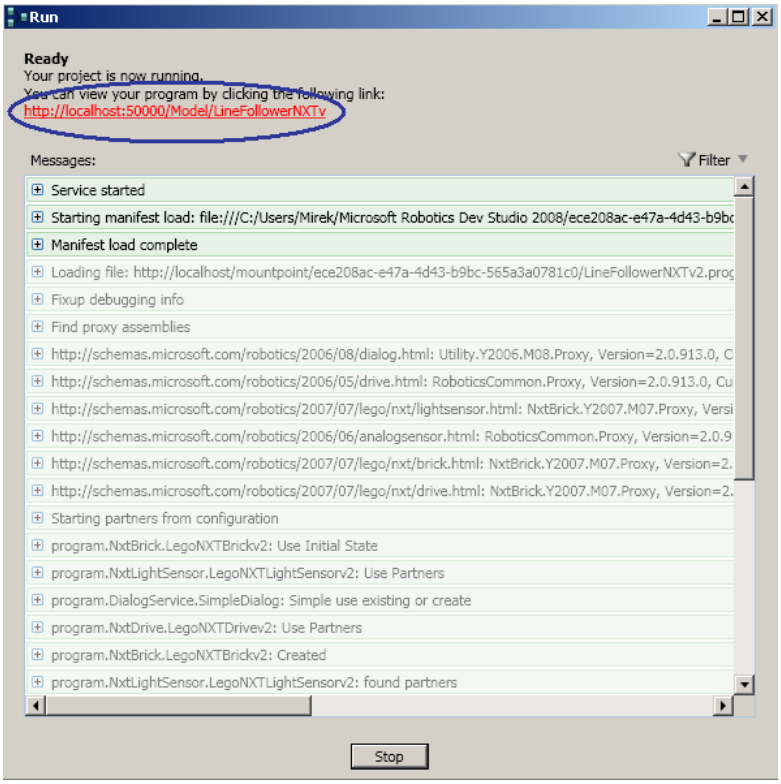
Obrázek 13: Prostředí Visual Programming Language

Seznam dostupných služeb můžeme najít na seznamu vlevo (viz obrázek č. 13). Služby se v tomto seznamu objeví až po jejich kompilaci, kdy se knihovna služby uloží do adresáře `...\Microsoft Robotics Dev Studio 2008\bin`. V tomto seznamu jsou služby jak pro reálného robota, tak pro simulační prostředí.

Program spustíme příkazem *Start*, který najdeme v nabídce Run v menu. Následně se v okně na obrázku č. 15 objeví seznam spuštěných služeb, jejich stav a případné problémy a chyby. Debugging ve VPL není typický. Služba Log poskytuje zaznamenávání proměnných během programu. Po spuštění a proběhnutí programu lze dané hodnoty najít přes odkaz v dialogu Run (obrázek č. 15). Dostaneme se až do debug and trace zpráv, kde je můžeme různě organizovat, číst a program debugovat.

4.3 Vytvoření nové aktivity

Novou aktivitu vytvoříme přenesením komponenty aktivity do diagramu. Následuje její nastavení. Dvojklikem otevřeme diagram nové aktivity. V diagramu tvoříme program. Nastavení vstupních, výstupních hodnot a notifikace se provádí v nabídce, kterou otevřeme pomocí ikony nahoře v liště nad diagramem aktivity (obrázek č. 16). Pro každou aktivitu můžeme naimplementovat více akcí, které se mohou provést. Mezi akcemi se přepínáme také v liště aktivity.



Obrázek 14: VPL - dialog Run

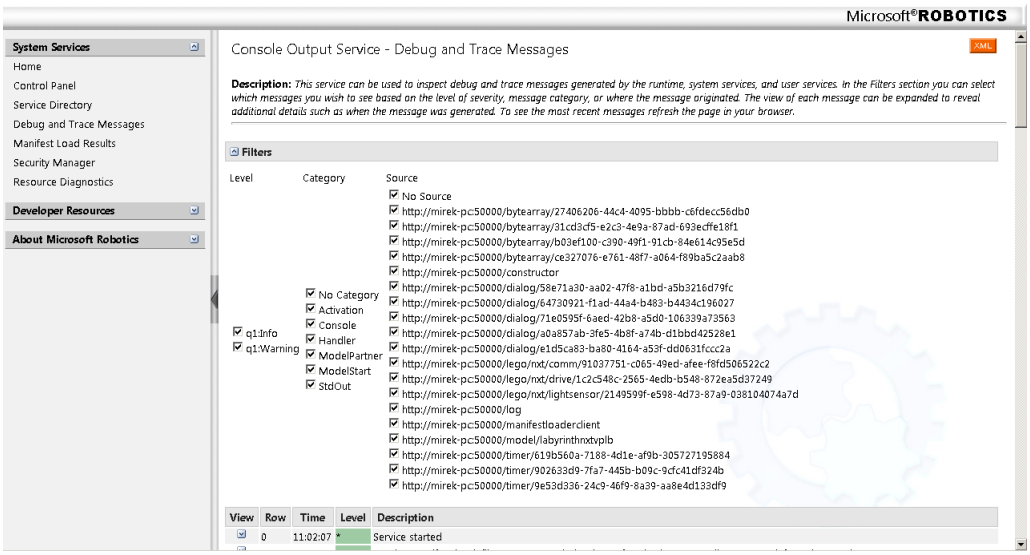
4.4 Kompilace programu do služby

Pokud chceme pro náš program vytvořit vlastní službu, je třeba ji mít zkompilovanou. Ve VPL toho docílíme příkazem *Compile as a service*, který najdeme v menu v Build. Do vámi nastaveného adresáře se program VPL zkompiluje do kódu C#, který následně lze otevřít, upravit a spustit v Microsoft Visual Studiu. Projekty MRDS musí být vždy umístěny ve složce `...\Microsoft Robotics Dev Studio 2008\`, jinak nebudou spustitelné. Bohužel, kód v C# z MRDS do VPL zkompilovat nelze.

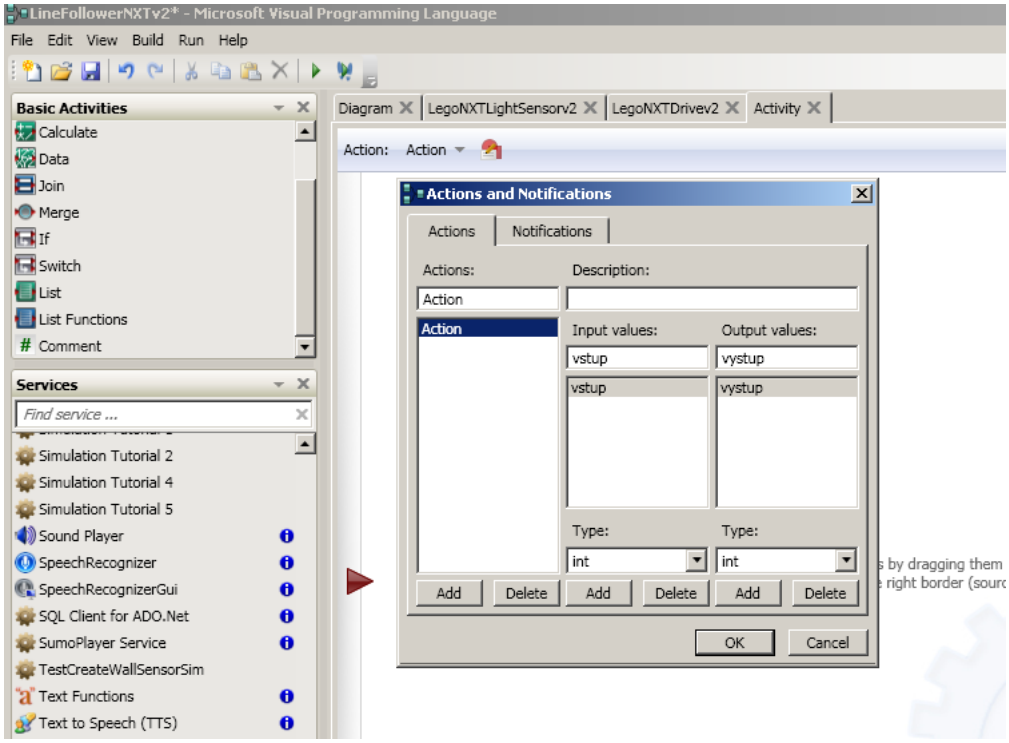
4.5 Omezení

Omezení VPL jsme zaznamenali při implementování úloh náročných na mnoho komponent. Jakmile se do diagramu umístilo okolo 70 komponent (služeb, aktivit apod.). VPL jen při otevření takového diagramu využívalo velkou část RAM paměti a začalo zamrzat a sekat se. Tímto se práce s ním stávala nepřijatelná.

VPL nemá defaultně k dispozici datový typ pole. Nicméně toto lze řešit použitím datového typu seznam. VPL nabízí službu ByteArray, pomocí které lze seznam přetypovat na pole a naopak. Práce s poli ve VPL je dosti neobratná, ale je možná.



Obrázek 15: VPL - Debug and trace zprávy



Obrázek 16: VPL - nová aktivita

5 Modelované úlohy

5.1 Konfigurace softwaru a hardwaru při implementaci

Některé modelované úlohy byly implementovány na reálném robotovi a některé ve VSE. K implementaci úlohy jsme používali sadu LEGO Mindstorms NXT(robot s firmware verzí 1.28) a notebook Hewlett-Packard Pavilion dv5 s následujícími parametry: procesor - Intel Core(TM)2 Duo P7350 - 2GHz a RAM paměť 2GB, grafický adaptér - NVIDIA GeForce 9200M GS.

Z hlediska software bylo pro implementaci použito Microsoft Robotics Developer Studio 2008 nad 32bitovým operačním systémem Windows Vista Home Premium, Service pack 2.

5.2 Popis dílčích úloh

První z úloh je jízda robota na nakloněnou rovinu. Tato úloha je implementována pouze ve VSE pomocí MRDS. Úloha demonstruje práci s entitami ve VSE, implementaci vlastního objektu pro VSE a práci s fyzikálním prostředím VSE. Úkolem robota je na nakloněnou rovinu vyjet, pohybovat se na ní a následně sjet.

Úloha překážka byla řešena jak pro VSE tak pro reálného robota v MRDS. Za pomocí využití dotykového senzoru robota musí robot při jízdě zjistit, že před ním stojí překážka a objet ji.

Další úlohou je jízda po čáře. Pro ní využíváme simulační pole s již připravenou čárou, které je součástí stavebnice. Robot za pomocí světelného senzoru rozpozná hranici bílé a černé barvy a na základě těchto inofrmací řídí pohyb. Implementovali jsme algoritmus tzv. *zig-zag* a následně jsme ho optimalizovali tak, aby pohyb byl plynulejší. Tato úloha je řešena jen pro reálného robota pomocí VPL.

Poslední úlohou je průjezd labyrintu. Labyrit je tvořen z černých čar ve čtvercové soustavě na světlém podkladě. Úkolem robota je na každé křižovatce zjistit možné cesty, celým labyrintem projet, najít nejkratší cestu ze startu do cíle a projet jí. Při této úloze je využito zejména světelného senzoru k rozpoznání odstínů barev. Úloha byla implementována jen pro reálného robota.

5.3 Nakloněná rovina

Pro úlohu jsme museli implementovat rovinu jako externí objekt, který jsme do MRDS přidali. Úkolem robota je na tuto rovinu vyjet a sjet a pohybovat se v simulovaném prostředí. Ve VSE můžeme měnit fyzikální podmínky pro robota (jeho sílu, gravitaci apod.) a následně sledovat jeho chování v dané situaci.

Prostředí zakládáme na již připraveném modelu z SimulationTutorial 1. Toto prostředí jsme upravili pro naše potřeby. Pro ovládání robota používáme partnerskou službu *simpledashboard*, která při startu simulace vyvolá tzv. *dashboard* - řídící desku.

5.3.1 Implementace

Celá implementace probíhala v kódu v MRDS a pomocí manifestů služeb. Projekt NaklonenaRovina jsme založili na SimulationTutorial1 a proto se nám do složky projektu nakopíroval i konfiguračni soubor *simulationengine.xml*, ve kterém je výpis a základní nastavení entit a fyziky v prostředí VSE.

Do manifestu *NaklonenaRovina.manifest.xml* přidáme tag <*ServiceRecordType*> s odkazem na partnerskou službu `simpledashboard`, která se spouští při startu simulátoru. Obsah manifestu je ve výpisu č. 2.

```
<?xml version="1.0" ?>
<Manifest
    xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html"
    xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html"
    >
  <CreateServiceList>
    <ServiceRecordType>
      <dssp:Contract>http://schemas.microsoft.com/robotics/2006/01/simpledashboard.html</dssp
          :Contract>
    </ServiceRecordType>

    <ServiceRecordType>
      <dssp:Contract>http://schemas.tempuri.org/2006/06/simulationtutorial1.html</dssp:Contract
          >
    </ServiceRecordType>
  </CreateServiceList>
</Manifest>
```

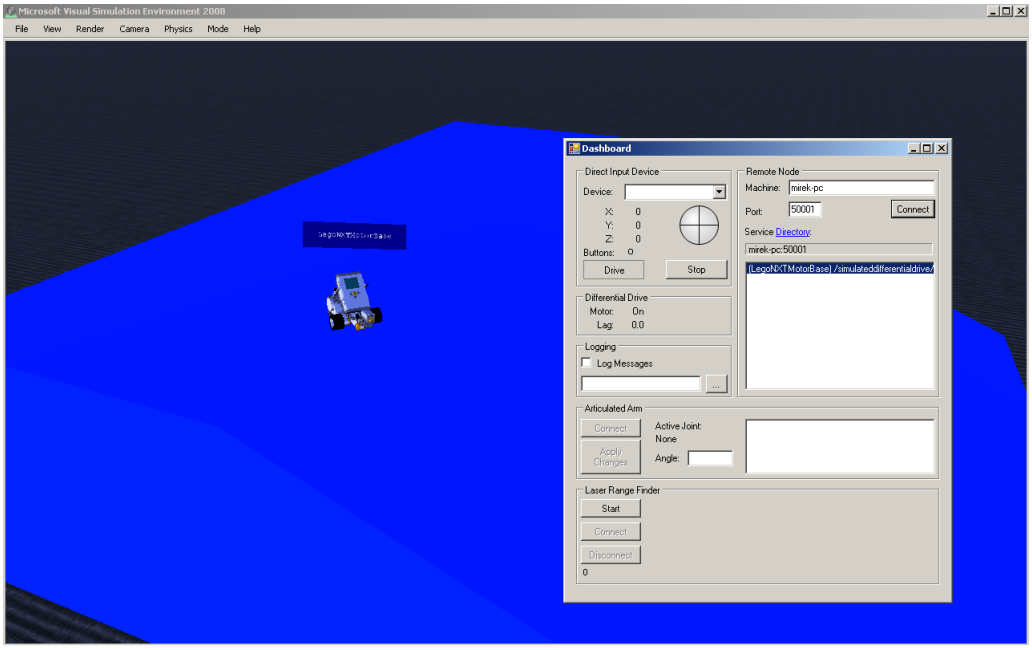
Výpis 2: NaklonenaRovina.manifest.xml

Nakloněnou rovinu jsme vytvořili ve freeware programu s názvem Anim8or v0.97b [13]. Tato utilita je pro tvorbu objektu dostatečná. Abychom mohli importovat do MRDS vlastní objekt, musí být ve formátu *.obj* (formát známý jako Wavefront formát, případně i s materiálovým souborem s příponou *.mtl*). V materiálovém souboru jsou uloženy informace o materiálu, ze kterého je těleso vytvořeno, jakou má barvu, povrch, drsnost, odstíny apod. Oba tyto soubory musí být uloženy v MRDS ve složce `..\Microsoft Robotics Dev Studio 2008\store\media`, ze které MRDS čerpá veškeré grafické modely, povrchy a další soubory k simulacím.

V kódu MRDS se již odkazujeme jen na soubor *.obj*. Simulátor při prvním načtení souboru *.obj* jej konvertuje na formát *.bos* a vytvoří s touto příponou nový soubor se stejným názvem. Při dalším spuštění simulace již MRDS pracuje s tímto souborem. Načítání souborů ve formátu *.bos* je pro MRDS rychlejší. Příkazový řádek MRDS obsahuje příkaz *Obj2Bos.exe*, kterým lze soubor zkonvertovat i bez spuštění simulátoru.

Při spuštění služby se zároveň spouští i řídící deska pro robota. Pro spojení desky a robota je třeba udělat následující kroky. V nabídce Remote Node v kolonce Machine napsat "localhost" a tlačítko Connect. Po tom, co se ve spodním okně objeví odkaz na NXT robota, je třeba na tento odkaz dvakrát poklikat (aby se Motor v sekci Differential Drive přepnul na On) a stisknout tlačítko Drive v sekci Direct Input Service. Teď lze jízdu robota jednoduše ovládat klikem a držením na řídící kolečko v sekci Direct Input service.

Ukázka simulovaného programu je na obrázku číslo 17.



Obrázek 17: Úloha - Nakloněná rovina

5.3.2 Problémy a poznatky k řešení

Pro správnou kompilaci projektu je třeba zkotrolovat správnou cestu k manifestu projektu. V properties projektu, záložka Debug v nabídce Start options, je třeba tyto cesty nastavit.

Při tvorbě externího objektu bylo náročné zjistit, že fyzikální vlastnosti materiálu, kterým je objekt pokryt musí být takové, aby robot po objektu opravdu vyjel. Dlouho se také stávalo, že robot materiál na objektu ignoroval a projel skrze stěnu do roviny. Tato vlastnost se na rovině objevila díky nekorektnímu exportu z programu Anim8or. Po upgradovaní programu byl export souborů .obj i .mtl v pořádku.

Při vkládání těles (entit) do VSE je třeba věnovat pozornost umístění těles. Jakmile se tělesa přerývají, pak je VSE “vystřeluje” od sebe podle sil a hmotnosti entit.

5.4 Překážka

Hlavním úkolem robota je rozpoznat překážku, na kterou během jízdy narazí a objet jí. Úlohu jsme řešeli jak pro simulátor, tak pro reálného robota. Využíváme dotykový senzor a na základě informací z něj zjišťujeme, zda robota do něčeho narazil.

5.4.1 Implementace úlohy Překážka v simulátoru

Zvolili jsme možnost implementace ve VPL. Pro simulaci používáme generické služby pro senzory i motory robota, které nám VPL nabízí. Simulaci jsme postavili na základě manifestu *LEGO.NXT.Tribot.Simulation.Manifest.xml*. Pro všechny služby pro robota, jako je GenericContactSensor a GenericDifferentialDrive musíme v základní konfiguraci nastavit použití manifestu a výše zmíněný manifest importovat. Díky tomu se nám do složky nakopíruje manifest a soubor `lego.nxt.tribot.simulationenginestate (LEGO.NXT.Tribot.Simulation.Manifest).xml`, ve kterém je nastavení simulačního prostředí. V tomto souboru si nastavíme pozice kamer, světla, robota i tělesa. Čili v tomto manifest souboru nastavujeme statiku simulace a v diagramu VPL implementujeme pouze dynamiku simulátoru.

Na základě notifikace z dotykového senzoru se robot zastaví, vrátí o kousek zpět a danou překážku objede. Jelikož program je založen na několika vláknech, je potřeba pohyb regulovat. K tomu využívame vlastnost dokončení pohybu DriveDistanceStage a RotateDegreesStage. Tyto vlastnosti mohou přecházet mezi stavy InitialRequest > Started > Completed nebo InitialRequest > Started > Canceled. Pouze jedna operace může být prováděna na motoru a mění se její stav. Aby robot dokončil všechna otočení a jízdy, využíváme podmínky se stavem jízdy Completed. Jinými slovy, po dokončení jedné činnosti může pokračovat v jiné.

5.4.2 Problémy a poznatky k řešení v simulátoru

K řízení pohybu robota je třeba využívat vlastnosti DriveDistanceStage a RotateDegreesStage. Dokud jsme je nevyužívali, pak korektně robot nedokončil pohyb, protože jiné vlakno robotovi posílalo jiné instukce.

Optimalizace programu spočívá i v nastavení síly, hmotnosti a rychlosti robota. Důležitá je také hmotnost a síla tělesa, do kterého robot naráží. Pokud tyto vlastnosti nejsou v určité rovnováze pak se může stát, že ve VSE robot těleso potlačí nebo se při nárazu překlopí.

5.4.3 Implementace úlohy Překážka v reálném prostředí

I pro reálné prostředí jsme zvolili k implementaci VPL. Využíváme dotykový senzor stejně jako v simulátoru. Používáme služby pro senzory konkrétního robota, které MRDS nabízí. V našem případě se jedná o LegoNXTBrickv2, LegoNXTTouchSensorv2 a LegoNXTDrivev2. Jakmile je senzor ve stavu *stisknutý*, pak notifikace oznámí jeho změnu stavu a spouští se vlákno, které vykonává pohyby - objetí překážky. Do vlákna jsme přidali aktivity WaitForDriveCompletition a Timer, které vlákno uspávají tak, aby daný pohyb byl vždy dokončen a nebyl přerušen následující aktivitou. Robot stejně jako v simulátoru narazí na překážku, couvá o 25 cm zpět a objíždí ji.

5.4.4 Problémy a poznatky k řešení v reálném prostředí

Řízení pohybu - otáčení a jízdu provádíme pomocí vlastnosti DriveStage (obdoba DriveDistanceStage v simulátoru) a pomocí časovačů, které jsou nastavené tak, aby byla každá aktivita dokončena.

VPL nabízí také službu WaitForDriveCompletition, kterou jsme také využili. Nicméně jsou s ní někdy problémy a chová se, jakoby v programu nebyla a program nepočká na dokončení aktivity. Z těchto důvodů jsme zvolili kombinovavý způsob řízení pohybů pomocí služby WaitForDriveCompletition a časovačů (služba Timer).

5.5 Jízda po čáře

K realizaci této úlohy využíváme pole s černou čarou ze stavebnice s robotem. Úkolem robota je sledovat černou čáru a jet po ní. Robot se nesmí z cesty vychýlit, ani vyjet. Tato úloha je vypracována ve dvou verzích, přičemž první verze je základní a druhá je optimalizována - pohyb robota je více regulovaný.

5.5.1 Implementace

Pro implementaci jsme využili VPL. Algoritmus je založen na informacích, které poskytuje světelný senzor. Světelný senzor používáme v aktivním módu, kdy je rozsvícená LED. Hodnoty světelného senzoru můžeme používat buď tzv. *čisté* nebo *normalizované*. Čistá hodnota je schovaná za vlastností RawMeasurement a obsahuje procentuální hodnotu naměřeného světla. Normalizovaná hodnota je číslo datového typu double od 0 do 1 a můžeme jí použít pomocí vlastnosti NormalizedMeasurement. Je to normalizováná hodnota čisté hodnoty světla vzhledem k vlastnosti RawMeasurementRange, což je maximální rozsah měření čísté hodnoty, který můžeme nastavit.

V naší úloze používáme čisté hodnoty. Hodnotu odstínu, který senzor vidí, ukládáme do proměnné a podle podmínek řídíme pohyb robota.

Algoritmus je založený na principu zig-zag. Robot v podstatě sleduje hranu černé čáry s bílým podkladem. Smysl algoritmu lze popsat pseudokódem ve výpisu č. 3. V našem případě jsou hodnoty světla pro bílou cca 63 a černou 35. Rozdělíme spektrum na tři intervaly (0-41, 41-58, 58-100). Senzor vrací hodnotu světla s tolerancí několika procent, proto k hraničním hodnotám přidáváme 5 procent.

```
POKUD je světlo < 41 PAK otoč vpravo
JINAK
        POKUD je světlo < 58 PAK jeď rovně
        JINAK otoč vlevo
        konec podmínky
konec podmínky
```

Výpis 3: Pseudokód algoritmu sledování čáry

Tento algoritmus pro nás nebyl dostatečný, protože pohyb robota je třeba více regulovat. Světelný senzor při přechodu z černé na bílou hodnoty mění, a proto jsme do algoritmu přidali další podmínky tak, aby byl pohyb více plynulější a přirozenější. Spektrum

viditelnosti rozdělíme místo tří na pět intervalů (0-26, 26-33, 33-38, 38-58, 58-70, 70-100). Prakticky jsme jen přidali pohyb otoč lehce doprava a lehce doleva do předchozího algoritmu. Smysl je zřejmý z výpisu č. 4

```
POKUD je světlo < 26 PAK otoč prudce doprava
JINAK
        POKUD je světlo < 33 PAK otoč lehce doprava
        JINAK POKUD je světlo < 38 PAK jeď rovně
                JINAK POKUD je světlo < 58 PAK otoč lehce doleva
                        JINAK POKUD je světlo < 70 PAK otoč prudce doleva
                                konec podmínky
                        konec podmínky
                konec podmínky
        konec podmínky
konec podmínky
```

Výpis 4: Pseudokód optimalizováného algoritmu sledování čáry

5.5.2 Problémy a poznatky k řešení

Hodnoty odraženého světla od povrchu se mohou měnit, pokud realizujeme úlohu v jiném prostředí. Senzor není na začátku kalibrován a podle prostředí je ovlivňován i okolním světlem. V důsledku těchto vlastností se musí hodnoty podmínek změnit dle aktuálního prostředí, kde se úloha realizuje.

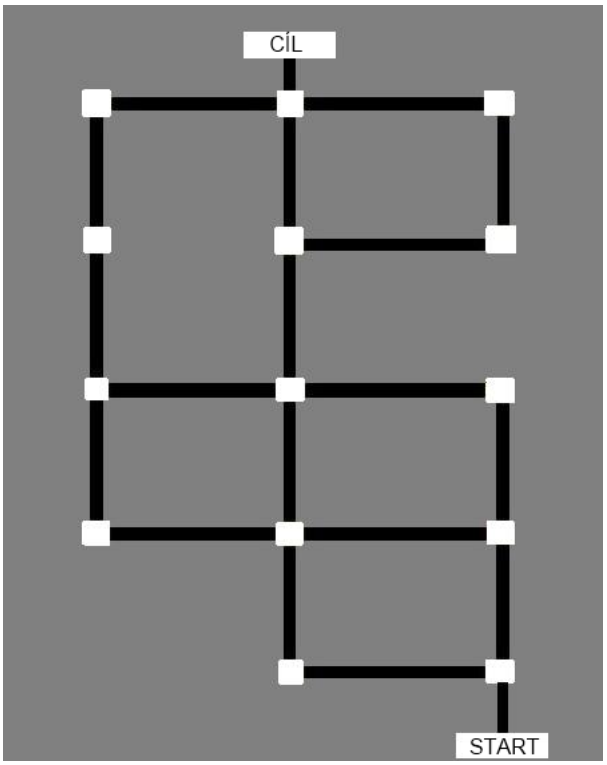
V průběhu řešení této úlohy jsme narazili na problém komunikace robota a MRDS přes bluetooth. Jelikož bylo potřeba rychlých reakcí na změny, které robot posílal, tak se zvýšil i počet zpráv, které se začaly ztrácet a robot se nechoval tak, jak bychom chtěli. V kapitole 3.3 je popsán způsob přenosu zpráv mezi robotem a počítačem. Robot při průměrné rychlosti jízdy nestíhal adekvátně reagovat na pokyny počítače. Tyto problémy nás nutily úlohu optimalizovat.

Optimalizaci jsme provedli metodou pokusů. Využili jsme snížení rychlosti otáčení robota do stran a úpravení hodnoty frekvence sběru dat z světelného senzoru a servo motoru. Tato frekvence určuje v jakém časovém intervalu bude zaslána zpráva o aktuálním stavu senzoru na počítač. Nastavuje se v konfiguračním okně daného senzoru. Defaultně je nastavena na hodnotu 0. V naší úloze jsme nakonec zvolili nastavení pro světelný senzor na 150 ms a pro servo motory na 80 ms.

Tato optimalizace by nebyla potřeba, kdybychom nepotřebovali komunikovat s počítačem přes bluetooth. Programy vytvořené v software pro LEGO NXT robota se nahrají přímo do řídící jednotky a odpadá problém se ztrátami zpráv. Jízdu po čáře lze pomocí tohoto softwaru realizovat plynuleji, rychleji a spolehlivěji než přes MRDS.

6 Labyrint

Cílem úlohy bylo navrhnout a realizovat průchod robota neznámým bludištěm a vyhledání nejkratší možné cesty skrz labyrint ze startu do cíle. Bludiště je čtvercová síť realizována černými linkami na světlé podložce. Vzor možného bludiště je na obrázku č. 18. Křižovatky jsou bílé čtverce na kříženích černé pásky tak, aby robot mohl rozpoznat, že dojel na křižovatku.



Obrázek 18: Vzor labyrintu

6.1 Návrh

Základem úlohy je jízda po čáře pomocí světelného senzoru. Jakmile robot vidí bílou barvu, ví, že se ocitá na křižovatce, kterou je třeba prohledat. Zjišťuje, jaké jsou možné cesty odjezdu z křižovatky a informace si ukládá. Start a cíl jsou reprezentovány silnějším bílým pruhem v bludišti tak, aby při prohledávání křižovatky robot uviděl třikrát bílou barvu místo očekáváné černé (hledání cesty před ním, vpravo a vlevo). Z těchto informací usoudí, kde je cíl.

Při jízdě a výběru další křižovatky používáme typický grafový algoritmus prohledávání do hloubky. Považujme křižovatky v bludišti za vrcholy v grafu. Základní myšlenkou prohledávání grafu do hloubky (tzv. Depth first search) je snaha postupovat stále

dál od počátečního vrcholu dosud neprozkoumaným směrem. Pokud dorazíme do vrcholu, ze kterého již nevede neprozkoumaná cesta, pak se vracíme k prvnímu možnému vrcholu, ze kterého vede alespoň jedna neprozkoumaná cesta a dále prohledáváme následující vrcholy (tzv. backtracking).

Pokud máme uložené informace o bludišti - jeho vrcholy a cesty, pak například podle Dijsktrova algoritmu necháme nalézt nejkratší cestu. Robota umístíme na výchozí pozici a robot projede bludiště až do cíle.

6.2 Postup řešení

V této úloze využíváme algoritmus popsaný v kapitole 5.5. Jakmile robot vidí bílou barvu, spouští se vlákno, na kterém probíhájí aktivity pro zpracování křižovatky - zapisování možných cest, sousedních vrcholů a další. Jediná informace, kterou robot zná před spuštěním programu, je počet křižovatek v bludišti, nicméně neví nic o umístění startu, cíle ani o struktuře bludiště.

Aplikace má vytvořen vlastní souřadnicový systém s x-ovou a y-ovou osou a vždy podle pohybu ukládáme aktuální umístění v bludišti. Dále označujeme vrcholy grafu = křižovatky bludiště, číslem. Než se robot začne pohybovat, máme určenou sadu směrů, což je reprezentace směrů z vrcholů čísly (nahoru - 1, dolů - 3, doprava - 2, doleva - 4). Uchováváme si i informaci o tom, o kolik stupňů je robot otočený, a podle toho se také sada směrů musí měnit, aby zůstaly zachovány stejná čísla pro spravné směry.

V programu pracujeme s maticí sousednosti vrcholů v grafu. Jakmile robot při jízdě uvidí bílou křižovatku, jako první se provádí kontrola souřadnic a čísla vrcholu. Zjišťujeme, jestli nejsme ve vrcholu, ve kterém jsme už byli, a pokud je vrchol nový, pak ho musíme prohledat a informace zaznamenat.

Podle daného směru a otočení se transformuje sada směrů. Robot dostane příkaz k prohledání křižovatky, což je realizováno jízdou vpřed o 7 cm tak, aby byl senzor nad cestou, a nasledným otočením o 90 stupňů vpravo a 180 stupňů vlevo, abychom zkontrolovali všechny směry. Informace o možných směrech zapisujeme pomocí čísel do matice směrů. Zároveň do této matice na první dvě pozice (index 0 a 1) zapisujeme souřadnice našeho systému pro každý vrchol (x-ová souřadnice na index 0 a y-ová souřadnice na index 1). Souřadný systém zavádíme, abychom se vyhli zacyklení robota. Jelikož je bludiště pravoúhlé, robot by mohl jezdit stále do čtverce aniž by věděl, že už vrcholy prošel. Souřadný systém toto zacyklení vylučuje. Matice *seznam_smeru* má počet řádků identický s počtem křižovatek v bludišti. Sloupce matice jsou x-ová souřadnice, y-ová souřadnice a následující 4 sloupce reprezentují možné směry vedoucí z vrcholu(nahoru, doprava, dolů, doleva). Do matice zapisujeme hodnoty 0, 1 a 2, které reprezentují informaci, zda cesta v daném směru neexistuje (0), je objevená (1) nebo je projetá (2). Při každém otočení robota zapisujeme do proměnně *orientace* hodnotu 0 - 360. Hodnota 0 stupňů přísluší jizdě v před, 90 stupňů otočení vpravo atp.

Po zkontrolování křižovatky se robot rozhoduje kudy pojede k dalšímu vrcholu. Již máme zjištěno, které cesty z daného vrcholu vedou a jestli jsou prohledané nebo ne. Robot si podle algoritmu vybírá první možnou cestu s označením 1 (objevená, ale neprohledaná) a skrze ní jede k dalšímu vrcholu. Jakmile je určen směr odjezdu z vrcholu

je možné spočítat souřadnice pro další vrchol (standardně při jízdě nahoru, resp. dolů se přičítá, resp. odečítá 1 na ose y a při jízdě doprava, resp. doleva se přičítá, resp. odečítá 1 na ose x). Tyto souřadnice se při příjezdu na další vrchol kontrolují přes matici směrů, abychom zjistili, jestli robot v tomto vrcholu ještě nebyl. Postupné směry odjezdů z křižovatek si ukládáme do seznamu pro případný backtracking.

Tento postup opakujeme tak dlouho až narazíme na vrchol, ze kterého již není možno odjet po cestě označené 1. V tom případě se podle algoritmu prohledávání grafu do hloubky vrátíme na první možný vrchol, ze kterého taková cesta vede. Tuto křižovatku nalezneme pomocí tzv. *backtrackingu* v seznamu směrů odjezdu z křižovatek. Seznamem budeme procházet od posledního prvku a směry transformovat na opačné, tím se dostaneme až k požadovanému vrcholu. Z tohoto vrcholu opět pokračujeme v prohledávání bludiště. Ve chvíli, kdy jsou všechny vrcholy matice vyplněné a všechny cesty označené číslem 2 (objevené a projeté), pak je bludiště prohledáno. Pomocí Dijkstrova algoritmu nalezneme v matici sousednosti nejkratší možnou cestu bludištěm a cestu uložíme pomocí směrů do seznamu, ze kterém při následné jízdě postupně čteme směry dojezdu z křižovatek.

6.3 Implementace a poznatky

Celá úloha je řešena ve VPL. Pro jízdu po bludišti využíváme algoritmus jízdy po čáře. Ve VPL máme jednotlivé části programu rozděleny do nových aktivit. Program obsahuje aktivity *jizda*, *zpracovani*, *krizovatka*, *prevod_indexu*, *zapis_do_seznamu* a *novy_seznam*. Většina těchto aktivit má naimplementovány i svoje akce, například *jízda* má akce *Otoc*, *Transformace_SadySmeru* a *Transformace_SmerNaStupne*.

VPL neposkytuje jako datovou strukturu jednorozměrné ani dvourozměrné pole. Nicméně pro naši úlohu jsme potřebovali pracovat s maticí. Jednorozměrné pole lze ve VPL realizovat pomocí služby *ByteArray*, která umí konvertovat seznam čísel datového typu byte na jednorozměrné pole a naopak. S jinými poli ve VPL pracovat nelze. Naše matice jsme museli reprezentovat jednorozměrným polem tak, že jsme jakoby poskládali řádky za sebe, toto vše jsme uložili do seznamu a následně převedli pomocí služby *ByteArray* na pole. Z tohoto pole lze klasicky číst prvky dle indexu apod.

Zápis prvku do matice, resp. seznamu je řešen specifickým způsobem. Nejdříve je třeba pole převést na seznam a prvek na daném indexu smazat a pak na daný index požadovanou hodnotu zapsat. Všechny tyto operace jsou prováděny nad aktivitou *List Functions*. Pokud jsou všechny úpravy na seznamu dokončeny je opět zapotřebí seznam konvertovat na pole bytů.

Při realizaci jsme se opět museli potýkat s problémy komunikace robota a počítače přes bluetooth. Opět jsme museli řešit rovnováhu posílaných zpráv pomocí frekvence sběru dat na světelném senzoru a motoru. Aby robot dostatečně rychle reagoval na změnu barvy je hodnota frekvence sběru dat na světelném senzoru vyšší než u jízdy po čáře.

Během implementace jsme využívali dvě různé aktivity *merge* a *join*. Rozdíl není zřejmý. Nicméně vyplývá z fuknčnosti a ovládání vlákna. *Merge* spojuje datové toky a jakmile přijdě na spoj alespoň jeden z nich, pak prochází dál. *Join* spojuje také datové toky, ale čeká až na spoj dorazí všechny vstupní toky a následně posílá impulz dále.

Tato úloha by se standardně neřešila pomocí vláken, ale sekvenčním programováním. Nicméně práce ve VPL je založena na vláknech. Přepínání mezi vláknem jízdy po čáře a vláknem, které řeší zápis a zpracování křižovatky řídí pomocná proměnná typu bool.

6.4 Nevyřešené problémy

V průběhu práce ve VPL jsem došli k závěru, že VPL je možno používat pro aplikace jednodušší na datové struktury a služby bez složitějších algoritmů. V diagramu, který obsahuje více než 70 komponent, si VPL zabírá velké množství RAM paměti a práce v něm se stává nerealizovatelnou. Naše úloha byla realizována i pomocí vlastních aktivit s akcemi. Počet komponent v hlavním diagramu se snížil, ale ne na realizovatelné množství.

Realizací v reálném prostředí nám vyplynuly i fyzikální aspekty úlohy. Vzhledem ke komunikaci přes bluetooth nebyl robot schopen při otáčení na křižovatkách reagovat dostatečně rychle na černou čáru, kterou viděl. Proto jsme otáčení řídili počtem stupňů, vzhledem k tomu, že bludiště je pravoúhlé, tak by to neměl být problém. Nicméně robot na křižovatku mnohdy nepřijede otočen přesně rovně, jak bychom chtěli a tím pádem i následné otačení je chybné. Toto vychází z algoritmu zig-zag při jízdě po čáře, kdy robot sleduje hranu mezi bílou a černou a může na křižovatku dojet lehce natočený jiným směrem. Tento problém by mohl být eliminován pomocí senzoru kompasu, případně využití dvou světelných čidel pro detekci čáry a tím pádem i správného natočení robota.

Vzhledem k vyvstalým problémům a fyzikálním chybám, které jsme nemohli odstranit (např. vyjetí robota z čáry, jak je popsáno výše), jsme se rozhodli práci ukončit ve fázi realizace prohledání bludiště. Pro korektní vyřešení úlohy by musel robot být vybaven dalším senzorem, což by vyžadovalo implementaci nových aktivit v diagramu VPL pro práci se službou daného senzoru. Pro úplné dokončení úlohy tj. realizaci vyhledávání nejkratší cesty v matici sousednosti by pak stačilo jen vytvořit novou službu, která by realizovala algoritmus pro vyhledání nejkratší cesty (např.: Dijkstrův algoritmus).

7 Závěr

V práci jsme se seznámili s multiplatformním nástrojem pro programování robotických aplikací MRDS. V první části se zabýváme teoretickým rozborem technologií a popisem struktury MRDS. Popisujeme i simulátor VSE, ve kterém implementujeme dvě z úloh.

Lépe jsme poznali práci v grafickém nástroji MRDS, ve VPL. I jeho struktura je v práci představena. V praktické části jsme popsali problémy s implementací úloh ve VPL.

Navrhli jsme několik úloh, které jsme realizovali pomocí robota LEGO NXT. Popsali jsme funkce a použití všech základních senzorů, které jsou dodávány společně s robotem ve stavebnici LEGO NXT Mindstorms.

Práci v simulátoru jsme demonstrovali na úloze *Nakloněná rovina* a *Překážka*, kterou jsme celou namodelovali a rovněž jsme ji realizovali v reálném prostředí. Práce popisuje postup implementace a problémy, se kterými jsme se setkali.

Ostatní navržené úlohy představily práci ve VPL a MRDS. Narazili jsme na úskálí komunikace robota a počítače pomocí bluetooth. Kvůli těmto problémům jsou mnohé úkoly pro robota problémem. Výhodou MRDS je možnost programování různých typů robota. Nicméně za mínus můžeme považovat nemožnost konverze kódu z MRDS a přímé nahrání do robota. Software dodáváný s robotem (jazyk NXT-G) umožňuje nahrát program přímo do robota, a tím odpadají problémy s komunikací. Robot reaguje na podněty podstatně rychleji a úlohy jsou efektivnější.

Práce na úloze *Labyrint* byla ukončena dříve, jelikož jsme narazili na problém s prací ve VPL s mnoha komponentami. Tento nástroj není určený pro implementaci programů, ve kterém je třeba využívat mnoho komponent. Dále vznikaly nepřesnosti v pohybu robota díky fyzikálním aspektům úlohy.

Zajímavým námětem na další řešení by mohla být realizace úlohy labyrint pomocí více senzorů, které již nejsou obsahem základní stavebnice (např. kompas, senzor barvy nebo senzor naklonění). Všechny úlohy byly realizovány jen s robotem LEGO NXT. Přínosem ve zkoumání programování v MRDS by se mohla stát i realizace jedné identické úlohy na robotovi LEGO NXT a jiném typu robota. Následné porovnání komunikace a implementace může přinést nové poznatky a výsledky.

8 Literatura

[1] Morgan, Sara, *Programming Microsoft Robotics Studio*, Washington: Microsoft Press, 2008.

[2] Johns, Kyle, Taylor, Trevor, *Professional Microsoft Robotics Developer Studio*, Indianapolis: Wiley Publishnig Inc., 2008.

[3] *LEGO© MINDSTORMS User Guide*, The LEGO Group., 2007.

[4] Gasperi, Michael, Hurbain, Philippe, Hurbain, Isabelle *Extreme NXT: Extending the LEGO MINDSTORMS NXT to the Next Level*, California: Apress, 2007.

[5] *Josefnav.cz [online]*. Dostupný z WWW: http://www.josefnav.cz/index.htm [cit. březen 2010]

[6] *FrankPr's World of Devices [online]*. Dostupný z WWW: http://blogs.msdn.com/frankpr [cit. březen 2010]

[7] *Wikimedia Commons [online]*. Dostupný z WWW: http://commons.wikimedia.org/wiki/File:3D_coordinate_system.svg [cit. 7.4.2010]

[8] *Lego camp 2007 [online]*. Dostupný z WWW: http://engk12.ece.missouri.edu/LegoCamp/Gallery.html [cit. 2010-04-07]

[9] *RobotShop [online]*. Dostupný z WWW: http://www.robotshop.ca/ProductSearch.aspx?qs=nxt [cit. 2010-04-07]

[10] *Wikipedie [online]*. Dostupný z WWW: http://cs.wikipedia.org/wiki [cit. 2010-04-07]

[11] *Tel Aviv University - Analysis of the NXT Bluetooth-Communication Protocol [online]*. Dostupný z WWW: http://www.tau.ac.il/∼stoledo/lego/btperformance.html [cit. 2010-04-08]

[12] *Technologie NVIDIA PhysX [online]*. Dostupný z WWW: http://www.nvidia.com/object/physx_new.html [cit. 2010-04-28]

[13] *Anim8or [online]*. Dostupný z WWW: http://www.anim8or.com/main/index.html [cit. 2010-04-28]

A Obsah CD

- text práce
- zdrojové soubory s úlohami - složka `ULOHY\ULOHA_Cislo_NazevUlohy`
- software pro modelování objektů - Anim8or
- zdrojové soubory - model nakloněné roviny - složka Anim8or